



C.) Amendments to the Specification:

Substitute the appended Specification appended hereto with the Substitute AMENDED version being the mark up version. A clean copy version is also appended.

D. AMENDMENT TO THE DRAWINGS - Substitute the appended formal drawings.



**Branch Prediction Apparatus and Process for Restoring Replaced Branch  
History for Use in Future Branch Predictions for an Executing Program**

**This invention deals with novel process and novel apparatus features which may be embodied in a single chip processor for significantly improving processor performance by enabling the restoration of branch predictions previously lost in a branch history table.**

**TECHNICAL FIELD**

**The present invention generally deals with increasing program execution performance by processor semiconductor logic chips. The improvement is obtained by uniquely preserving and enabling the reuse of branch history in a branch history table (BHT) for associated instructions replaced in an instruction cache (I-cache) of a processor. Prior branch prediction techniques using branch history tables have lost the branch history associated with instruction lines replaced in an instruction cache.**

**RECEIVED**

**AUG 03 2004**

**Technology Center 2100**

**INCORPORATION BY REFERENCE:**

**Incorporated by reference herein is the entire specification, including all disclosure and drawings, of application having USPTO serial number 09/436264 filed on November 8, 1999 entitled "Increasing the Overall Prediction Accuracy for Multi-Cycle Branch Prediction Processes and Apparatus by Enabling Quick Recovery" invented by the inventor of the present application, now US Patent 6598152, granted July 22, 2003.**

**BACKGROUND**

**In prior art computer systems using branch history tables (BHTs), each BHT entry contains fields that predict the taken or not taken branch path for each branch instruction in an associated line of instructions in an instruction cache (I-cache). Each line of instructions contains N number of instruction locations, and each of the N instruction**

1 locations may contain any type of instruction, e.g. a branch instruction or a non-branch  
2 instruction. There are N number of BHT fields in any BHT entry respectively associated  
3 with the N instruction locations in the associated I-cache line. Each BHT field may be  
4 comprised of one or more bits, and is sometimes referred to as a counter field. In the  
5 detailed example described herein, each BHT field comprises a single bit.

6 Any distribution of instruction types may exist in any I-cache line. Accordingly, a line of  
7 instructions within any I-cache entry may contain no branch instruction, or any  
8 combination of branch and non-branch instructions. For example, each I-cache entry may  
9 comprise an instruction line with 8 instruction locations, and each of these eight instruction  
10 locations may contain an unconditional branch instruction, a conditional branch  
11 instruction, a non-branch instruction, or any other type of instruction. Thus, any  
12 distribution of instruction types may exist in any I-cache line. For example, the I-cache  
13 may have 32 K line entries. The I-cache index locates both an I-cache entry in the I-cache  
14 and an associated BHT entry in the BHT. Further, each BHT entry contains 8 BHT fields  
15 (e.g. bits) which are respectively associated with the 8 instruction locations in the associated  
16 I-cache entry. The only BHT bits in the BHT entry which are predictively effective are  
17 those associated with a branch instruction location, and the BHT bits associated with  
18 instruction locations containing non-branch instructions are ignored. For example, a BHT  
19 entry having a BHT bit set to a "1" state is predicting that a branch instruction in its  
20 associated location will be "taken", i.e. jump to a non-sequential instruction location on its  
21 next execution in the program. A "0" state for this BHT bit predicts its associated  
22 conditional branch instruction will be "not taken", i.e. go to the next sequential instruction  
23 location in the program. A BHT bit associated with an unconditional branch instruction is  
24 always set to the "1" state to indicate it is always "taken". The state of a BHT bit  
25 associated with a non-branch instruction is ignored, regardless of whether it has a "1" or  
26 "0" state.

27 In the prior art, a new line of instructions may be fetched from an L2 cache into an I-cache  
28 entry and replace a line of instructions previously stored in that I-cache entry. However,

1 the BHT entry associated with that I-cache entry is not replaced in the BHT when the  
2 instruction line is replaced in the I-cache entry. Whatever BHT prediction states exist in  
3 the BHT entry are assumed to be the predictions for the branch instruction(s) in the newly  
4 fetched line, even though the new line probably has branch instructions in different  
5 locations than the replaced I-cache line, and even though the existing BHT predictions may  
6 have been generated for other branch instructions in the program. Hence, the BHT  
7 predictions for a replaced line have a significant chance of providing wrong predictions for  
8 the branch instructions in the line.

9 When a BHT prediction selects the wrong branch path in the program, a sequence of  
10 incorrect instructions are selected and executed, because the selection of the wrong branch  
11 path is not immediately detected, but is detected many instruction execution cycles later.  
12 After detection, instruction results for these wrong instructions are destroyed, and the  
13 branch path is belatedly reset to the correct branch path from which the program  
14 execution continues, and the wrong BHT branch prediction is corrected in the BHT.  
15 Hence, wrong BHT predictions may cause significant time loss during program execution  
16 due to their selection of incorrect branch paths. This increase in the program execution  
17 time causes a corresponding reduction in the processing rate of executing programs. The  
18 resetting of wrong branch paths and the correction of BHT erroneous predictions is taught  
19 in the US Patent 6598152, granted July 22, 2003.

20 The statistical probability of BHT predictions being incorrect for a replaced line is  
21 variable. For example, if a newly fetched instruction line replaces a branch instruction  
22 with an unrelated branch instruction in the same I-cache location, the existing setting of its  
23 location associated BHT prediction is expected to have a 50 percent probability of being  
24 correct (and a 50 percent chance of being wrong). But if the new branch instruction in the  
25 newly fetched line replaces a non-branch instruction, and if this newly fetched instruction  
26 was the last branch instruction previously in that instruction location, its  
27 location-associated BHT prediction has better than a 90 percent probability of being  
28 correct.

1 In the known prior art of BHT branch prediction techniques, the predictions in the branch  
2 history table were lost when associated branch instructions were replaced in the I-cache.  
3 The subject invention may be used with some of these prior BHT branch prediction  
4 systems to improve their BHT prediction rates.

## 5 SUMMARY OF THE INVENTION

6 This invention increases the speed at which a processor can execute a program by  
7 increasing the accuracy of its BHT branch predictions. This increases the processing  
8 speed of a program (even when there is no change in the instruction execution cycle time of  
9 the processor) by preventing the loss of previously-generated BHT predictions (which were  
10 lost in the prior art after replacement of associated branch instructions in the I-cache). For  
11 example, this invention may increase the BHT branch prediction accuracy for a branch  
12 instruction refetched to the same location in an I-cache entry - by increasing its probability  
13 of correctness from a potential 50 percent rate to in excess of a 90 percent rate. This is  
14 better than an 80 percent improvement in the prediction accuracy for branch instructions  
15 refetched in an I-cache, i.e. computed as  $(90-50)/50 = 80$ .

16 When an I-cache line of read-only instructions is replaced in an I-cache, there is no castout  
17 of the replaced line because it has a copy available in the storage hierarchy for being  
18 refetched later into the I-cache. Also associated with that I-cache instruction line is a BHT  
19 entry which is not castout but may contain predictions that do not correctly predict the  
20 "taken or not taken" outcome of one or more branch instructions in the refetched line.

21 With this invention, when a line of instructions is replaced in the I-cache, the current state  
22 of its associated BHT entry is stored in a hint instruction in the I-cache. Later, the hint  
23 instruction is stored in the system storage hierarchy in association with a copy of the  
24 replaced I-cache instruction line. Also stored in that hint instruction are: a branch mask  
25 indicating the locations of any branch instructions within the replaced I-cache line.

1 In the detailed embodiment described herein, an associated hint instruction is generated  
2 and stored in the I-cache when the associated line is accessed therein. When the line is  
3 later replaced in the I-cache, its hint instruction is then stored in a second level cache in  
4 association with a copy of the replaced I-cache instruction line. This invention may be used  
5 in hierarchy levels below the second level cache, such as a third level represented by the  
6 main memory of a system. When this invention is not extended to a third hierarchy level,  
7 the hint instruction is lost when its associated instruction line is replaced in the second level  
8 cache. Nevertheless, this invention is highly useful when it is only extended to the second  
9 level in the hierarchy, because line replacement in a large second level cache is rare.

10 Extension to one or more additional storage levels is an economic tradeoff, whereby the  
11 cost of extension to a next hierarchy levels may be outweighed by the low frequency of  
12 instruction line's refetches at the lower hierarchy levels involving only a very small increase  
13 in program execution efficiency, such a fraction of 1 percent. However, the subject  
14 invention comprehends the transfer and storage of hint instructions to one or more storage  
15 levels beyond the second level cache in the system storage hierarchy.

16  
17 In more detail, during an I-cache hit a hint instruction is generated and stored with its  
18 instruction line in a row of the I-cache to associate the hint instruction and the I-cache  
19 instruction line. When an I-cache miss occurs for the instruction line, the hint instruction  
20 is transferred from the I-cache to a row in the L2 cache containing the L2 copy of the  
21 associated instruction line. Then the I-cache line and its hint instruction are replaced by  
22 another instruction line and hint instruction copied from a row in the L2 cache located by  
23 the current instruction address (in IFAR). The replacing hint instruction will be a  
24 developed (generated) hint instruction if its L2 copy was previously used during the  
25 current execution of its program, i.e. the line is being fetched again (i.e. refetched) into the  
26 I-cache - then its associated hint instruction is fetched and used to restore predictions in the  
27 current BHT entry for branch instructions in the refetched line. This BHT entry  
28 restoration process does not affect its BHT bits corresponding to non-branch instructions  
29 in the refetched line. Thus, the restoration can only affect BHT predictability for branch

1 instructions in the newly fetched instruction line and does not affect the predictability of  
2 BHT bits associated with non-branch instructions in the associated instruction line. A  
3 “branch mask” in the hint instruction aids in the restoration by indicating the locations of  
4 any branch instructions in its associated instruction line.

5 Thus, the number of restored bit positions in a BHT entry is dependent on the number of  
6 branch instructions in the associated replaced line, and the branch instruction locations in  
7 the line are indicated by the branch mask in the hint instruction. If all instruction  
8 positions in a replace line contain branch instructions, all predictions in the associated  
9 BHT entry may be restored by this invention. But if less than all predictions in the  
10 associated BHT entry contain branch instructions, less than all BHT fields in the associated  
11 BHT entry are restored by this invention. Most instruction lines have less than all of its  
12 locations containing branch instructions, and some instruction lines have no branch  
13 instructions.

14 In the described embodiment, each hint instruction contains an operation code (op code) to  
15 identify a developed hint instruction, which contains a BHT index (bht\_index) that locates  
16 the associated BHT entry in the BHT, a branch mask (branch\_mask), and a BHT entry  
17 (bht\_bits) which stores a copy of the BHT entry having the BHT states existing when its  
18 associated instruction line was replaced in the I-cache. The branch mask has a “1” mask  
19 bit at each BHT field position associated with a branch instruction position in the  
20 associated instruction line. A “0” mask bit is provided at each branch mask position  
21 corresponding to a non-branch instruction position in the associated instruction line. In a  
22 restored BHT entry, the only changeable BHT positions correspond to the “1” positions in  
23 the branch mask. During the restoration process, each BHT field position in the BHT  
24 entry located at a corresponding “1” state mask-bit position is set to the state of the  
25 corresponding prediction position in the BHT field (bht\_bits) stored within the same hint  
26 instruction. In the BHT entry, no change is made to each BHT field position located by a  
27 “0” state mask-bit position.

1 Consequently, this invention allows the “0” mask bit positions in a restored BHT entry to  
2 represent predictions made for branch instruction(s) in different instruction lines that may  
3 later be refetched into the associated I-cache entry, as long as those branch instruction(s)  
4 are at non-branch locations in the currently replaced instruction line.

5 Accordingly, the process of this invention increases BHT prediction accuracy by enabling  
6 each BHT entry for a refetched instruction line to restore only the BHT predictions for the  
7 branch instruction positions in the refetched line. The avoidance of changing BHT  
8 predictions at non-branch instruction positions in a restored BHT entry has the useful  
9 benefit of allowing the non-branch BHT positions to retain predictions previously made for  
10 another instruction line that may in the future be refetched. This allows a restored BHT  
11 entry to retain predictions for multiple different instruction lines when such predictions  
12 are located at BHT positions which will not be used by any instruction in the currently  
13 associated line.

14 Novel apparatus is described in the detailed embodiment to support this inventive process  
15 by modifying both the I-cache and the second-level cache to receive and store hint  
16 instructions in association with instruction lines stored therein. This is done in both the  
17 first level I-cache and the second level cache by structuring each row in each cache to store  
18 both an instruction line and an associated hint instruction. The hint instruction location  
19 in each row is initialized by storing therein a “no operation” (NOP) type of hint instruction.  
20 This may be done by using a NOP code in the operation code field of a hint instruction and  
21 ignoring all other fields in the NOP instruction when it is detected as a NOP. The first time  
22 during a program execution an instruction line is fetched into the I-cache from the L2  
23 cache in response to a current cache miss, the accessed L2 cache row will have been  
24 initialized with a NOP hint instruction, and this instruction line and its NOP are copied  
25 into the I-cache row having the current cache miss. The NOP may contain all “0” states in  
26 its “branch\_mask” and “bht bits” fields to prevent any restoration in the associated BHT  
27 entry at this time. However, if this instruction line thereafter has an I-cache hit, a real hint  
28 instruction (in the form described above) is generated and stored over the NOP hint



1 instruction in the associated I-cache row. Later when this I-cache line has a miss, this real  
2 hint instruction is copied from the I-cache row to overlay the corresponding NOP hint  
3 instruction in the L2 cache row containing a copy of the instruction line having the cache  
4 miss. Then the line and hint instruction are replaced in that I-cache entry. Then during  
5 the continuing execution of the program, this L2 stored hint instruction is available to  
6 restore its associated BHT entry when and if its associated instruction line is refetched  
7 from the L2 cache into the I-cache. The restored BHT entry fields then have the benefit of  
8 using the latest prediction for their associated instructions, thus having a greater chance of  
9 representing a correct BHT prediction.

---

10 Hence, it is the primary object of this invention to reduce the occurrence of wrong BHT  
11 predictions for a program by the restoration of BHT predictions lost in an I-cache by  
12 replacement of instruction lines therein without affecting associated BHT predictions  
13 which cannot be currently used. The invention increases processor execution speed by  
14 expanding the amount of branch history available to an executing program beyond the  
15 prediction capacity of the BHT, and this invention makes the replaced branch history  
16 quickly available from another level of the system storage hierarchy for later use during  
17 execution of a program.

18 The restoration process of this invention may overlap the normal operation of standard  
19 cache operations so that little or no processor execution time need be lost when this  
20 invention is used.

21 This invention discloses and claims novel "hint instruction" micro-control processes and  
22 apparatus which can operate in parallel with the normal program instruction processing  
23 controls of a processor to enable BHT predictions for replaced branch history to be stored  
24 in a usable form at another level in a storage hierarchy from which it can be quickly  
25 retrieved and used by an executing program. The micro-controls disclosed and claimed as  
26 part of this invention are preferably embedded in, and part of, the same semiconductor  
27 chip that contains the processor executing the program. Novel "hint instructions" are

1 generated and used by the novel processes disclosed and claimed herein in these the  
2 micro-controls.

3 The hint instructions may operate transparent to a program executed with conventional  
4 program instructions, while hint instructions are being concurrently generated and  
5 executed by the "hint processing" micro-controls in the same chip as the processor  
6 executing the program.

7 Both an instruction line and an associated hint instruction may be stored in the same row  
8 of an L1 cache and an L2 cache. The L1 and/or L2 cache structure may be designed using  
9 separate subarrays, one subarray for storing the program instruction lines (i.e. in a  
10 "instruction cache" subarray), and the other subarray for storing the associated hint  
11 instructions (i.e. in a "hint instruction" subarray). This modified structure may have the  
12 advantage of enabling each subarray to have a bit width that is a power of 2, which is a  
13 design preference with some cache designers. Then the line index for selecting a line in the  
14 cache subarray would also be used to select the associated hint instruction in the "hint  
15 instruction" subarray. Part of the same IFAR address selects the BHT entry in a separate  
16 BHT subarray .

17 In the detailed embodiment described herein, the term "hint instruction cache (HIC)" is  
18 generally used to identify a novel I-cache in which each row stores both an instruction line  
19 and its associated hint instruction.

20 Thus, this invention provides a novel hint instruction having novel controls using novel  
21 hardware and novel processes, which enable the saving and fast utilization of branch  
22 history for instructions replaced in an I-cache - to store their branch history elsewhere in  
23 the storage hierarchy, which if lost would require the inefficient process of resetting more  
24 wrongly-selected branch paths and belatedly redeveloped BHT predictions to replace the  
25 lost BHT predictions.

## **1 BRIEF DESCRIPTION OF THE DRAWINGS**

**2 FIGURE 1 illustrates the hint instruction form used in the described embodiment of the**  
**3 subject invention.**

**4 FIGURE 2 shows an overview of instruction execution controls in a processor having the**  
**5 novel hint instructions and processes shown in the other FIGUREs for the detailed**  
**6 embodiment.**

**7 FIGURE 2A shows the hint processor represented in FIGURE 2. FIGURE 2B represents**  
**8 the hardware logic of the “new BHT creation logic” circuits in FIGURE 2A.**

**9 FIGURE 3 is a modified view of the hint instruction controls represented in Figure 2.**

**10 FIGURE 4 represents the branch information queue (BIQ), and the form of its queue**  
**11 entries shown in block form in FIGURE 3.**

**12 FIGURE 5 represents a branch history table (BHT) associated with the Hint Instruction**  
**13 Cache IL1 seen in the block diagram of FIGURE 2.**

**14 FIGURE 6 shows the general form of the novel hint instruction cache (IL1) and its**  
**15 instruction cache Directory (IL1 Dir) shown in FIGUREs 2 and 3.**

**16 FIGURE 7 shows the general form of a L2 Cache Directory and its novel associated L2**  
**17 Cache shown in block diagram form in FIGURE 2.**

**18 FIGURES 8, 9 and 10 are flow diagrams that include hint instruction processes according**  
**19 to the subject invention which operate during the execution of program instructions for**  
**20 extending branch-history predictive operations in a branch history table (BHT).**

1 **FIGURE 11 shows a flow diagram of an Instruction Decode and Dispatch subprocess used**  
2 **in FIGURE 9.**

3 **FIGURE 12 shows a flow diagram of an Instruction Issue and Instruction Execution**  
4 **subprocess used in FIGURE 9.**

5 **FIGURE 13 shows a flow diagram of the subprocess performed by the hint processor**  
6 **shown in FIGURE 2A.**

## 7 **DESCRIPTION OF THE DETAILED EMBODIMENT**

8 **The detailed embodiment described herein has novel processor hardware shown in block**  
9 **form in FIGUREs 2, 2A, 3, 4, 5, 6 and 7, which may be structured on a processor chip, and**  
10 **FIGURES 8, 9, 10, 11, 12 and 13 represent the detailed novel process and novel**  
11 **subprocesses performed by the illustrated hardware.**

12 **FIGURE 2 includes a novel hint instruction cache (IL1) 201 and a novel L2 cache 212, each**  
13 **capable of containing a multiplicity of novel hint instructions, and conventional program**  
14 **instructions. Program instructions are fetched from the L2 cache 212 into the instruction**  
15 **cache (IL1) 201 for execution by the program currently being executed in the processor.**  
16 **The hint instructions in L2 cache 212 and in IL1 201 are each located in a respective row**  
17 **containing a line of instructions. In each cache, an association is obtained between an**  
18 **instruction line and a hint instruction by their being placed in the same cache row.**

19 **Either real addresses, or virtual addresses translated in the conventional manner by the**  
20 **processor, may be used by the executing program to address program instructions and**  
21 **data in a main storage of the processor system, and in each of the caches through their**  
22 **respective cache directories in the conventional manner. Any size virtual addresses may be**  
23 **used, such as 64 bit or 32 bit addresses.**

1 **FIGURE 1 shows the form of each hint instruction 100 and NOP hint instruction 109**  
2 **stored in caches 201 and 212 in FIGURE 2. The hint instructions are each shown as a 32**  
3 **bit instruction. The hint instructions may operate within the processor in a manner**  
4 **transparent to the executing program.**

5 **NOP (non-operational) instruction 109 is used for initializing the space to be occupied by a**  
6 **hint instruction 100, and the NOP format contains only the NOP code in the first 5 bits of**  
7 **the instruction and its remaining bits are unused. Hint instruction 100 has a load BHT**  
8 **operation code in its five bit positions 0-4 labeled "ld\_bht op". The NOP instruction type**  
9 **is used in the described embodiment to initialize storage space which later may be filled**  
10 **with the "ld\_bht op" load " hint instructions. In this embodiment, the load BHT hint**  
11 **instruction and the NOP instruction are each 4 bytes long, i.e. 32 bits. The length of each**  
12 **field in these instructions are indicated by dimension arrows in each of the instructions in**  
13 **FIGURE 1, and each dimension arrow is labeled with a centered bit number to indicate the**  
14 **bit length of its respective field. Thus, instruction 100 includes the five bit "ld\_bht op"**  
15 **field, an eleven bit "bht\_index" field, an eight bit "branch mask" field (br\_mask), and an**  
16 **eight bit "bht\_bits" field. As previously stated, the "ld\_bht op" field is the operation code**  
17 **of instruction 100. The bits in the "bht\_index" field provide the 48:58 index to locate and**  
18 **associate an IL1 cache entry (containing an instruction line), its IL1 directory entry, and**  
19 **their associated BHT entry. The "branch mask" field contains 8 bits, and each branch**  
20 **mask bit corresponds to a respective one of the 8 instruction locations in the associated**  
21 **instruction line. A mask bit is set to the "1" state to indicate when its respective instruction**  
22 **location contains a branch instruction, and is set to the "0" state to indicate when its**  
23 **respective IL1 instruction location does not contain a branch instruction. The "bht\_bits"**  
24 **field stores the content of a BHT entry located at the "bht\_index" in the BHT for the BHT**  
25 **entry associated with an instruction line being replace in the IL1 cache.**

26 **Each hint instruction is generated and stored in the hint instruction location identified by**  
27 **the current IFAR address, when the associated instruction line in IL1 cache 201 is being**  
28 **accessed with a cache hit.**

1 A hint instruction is executed when its associated instruction line has a cache miss in the  
2 IL1. Then, the associated hint instruction is used to change the associated BHT entry if the  
3 associated instruction line has any branch instruction(s). The change in the associated  
4 BHT entry is only at a BHT bit located at a “branch mask” bit position having a “1” state  
5 (indicating the corresponding instruction is a branch instruction), if the “branch mask”  
6 has any “1” bit. Then, only the “1” mask bit position(s) are located in the current BHT  
7 entry where they are set to the “1” or “0” bit state of the corresponding bit position in the  
8 “bht\_bits” field of the hint instruction, i.e. only at the “1” mask bit position(s) in the BHT  
9 entry. The “0” mask bit locations in the associated BHT entry are not affected by the  
10 process of executing the associated hint instruction.

11 During an IL1 cache miss, the associated hint instruction stored in the IL1 cache 201 is  
12 copied to the L2 cache immediately before the associated instruction line in the IL1 cache  
13 201 is overlayed in the IL1 cache by a new instruction line fetched from the L2 cache. The  
14 L2 location for this hint instruction is generated from the content of the associated IL1  
15 directory entry, i.e. from a “address of the first instruction” field that indicates the address  
16 of the first instruction to be executed in the associated instruction line.

17 Generally, a NOP instruction marks an entry location in the L2 cache which does not  
18 contain any IL1 replaced entry. That is, a NOP indicates an L2 entry which may contain a  
19 copy of an instruction line that have not been replaced in IL1 201, although it may have  
20 been copied into the IL1 cache where it currently exists. A NOP instruction is overlayed by  
21 a newly generated “Ld\_bht” instruction when its corresponding IL1 location is first used in  
22 the IL1 cache 201.

23 An IL1 index 48:58 is used to locate a row of IL1 instructions in IL1 201 and its  
24 corresponding IL1 directory entry in directory entry 202. The IL1 index is obtained from  
25 the eleven address bit positions 48 through 58 (i.e. 48:58) in IFAR 203 in FIGURE 2. The  
26 rows in the IL1 cache is shown divided into two sections 201A and 201B (FIGURE 3) which

1 respectively contain the instruction lines and the hint instructions. However, these sections  
2 may instead be obtained by using separate hardware subarrays which are accessed with  
3 the same index 48:58. The value of the 11 bits obtained from the sequence of address bit  
4 positions 48:58 locates one of 2048 rows in IL1 201 and also locates the corresponding row  
5 in IL1 directory 202 to associate the IFAR address with these two selected rows.

6 The IL1 index bits 48:58 of the IFAR address are also used to select a BHT entry in a BHT  
7 204, shown in FIGUREs 2 and 3. Thus, IFAR bits 48:58 associate a BHT entry in BHT 202  
8 with an instruction line and a hint instruction in IL1 201 and its corresponding directory  
9 entry in the IL1 directory 202.

10 IL directory 202 is a conventional and contains a "valid bit" field and a 48 bit "address of  
11 the first instruction" (i.e. first instruction address) field. A valid state in the valid bit field  
12 indicates that the associated IL1 row (in IL1 201) contains an instruction line, and that the  
13 "first instruction address" field locates the first program instruction to be selected for  
14 execution in that instruction line. An invalid state of the valid bit indicates the contents of  
15 the corresponding IL1 row are invalid and should not be used.

16 In this embodiment, each IL1 row contains space for eight 32 bit program instructions and  
17 one 32 bit hint instruction shown at the end of each row. Hence, each program instruction  
18 and each hint instruction has an instruction length of 4 bytes (i.e. 32 bits), in the respective  
19 columns 201A and 201B of IL1 201. In the IL1 directory row, each "first instruction  
20 address" field contains 48 bits which locate the first program instruction to be accessed in  
21 the corresponding row in IL1 201.

22 The first instruction address" field in the IFAR selected IL1 directory row is used only if  
23 the content of the "first instruction address" field in that row matches the current address  
24 bits 0:47 in IFAR 203. When a compare-equal occurs between IFAR bits 0:47 and the  
25 "first instruction address" field in the accessed IL1 directory row, the addressed first  
26 instruction is allowed to be used in the associated IL1 row.

1 In FIGURE 2, the BHT 204 operates with IL1 201 to provide a prediction of whether the  
2 branch instructions stored in IL1 201 are to be “taken” or “not taken” in the program  
3 being executed. Generally, a “taken” branch instruction indicates the instruction path is to  
4 go to the address indicated by that instruction, and a “not taken” branch instruction  
5 indicates the instruction path is to continue with next sequential instruction in the  
6 program.

7 Each BHT entry in this embodiment contains eight 1-bit prediction fields. The sequence of  
8 the eight 1-bit prediction fields in any BHT row respectively provide a prediction of the  
9 “taken” or “not taken” state for each branch instruction at the corresponding position in  
10 the line. A BHT bit is ignored when its corresponding program instruction is not a  
11 conditional branch instruction. Thus, the only meaningful prediction bit(s) in any BHT  
12 row are those that correspond to a conditional branch instruction in the associated IL1  
13 row. The 0 state of a BHT prediction bit indicates it is predicting the “not taken” state for  
14 any corresponding conditional branch instruction, and the 1 state of a prediction bit  
15 indicate it is predicting the “taken” state for any corresponding conditional branch  
16 instruction.

17 FIGURE 3 shows in more detail parts of FIGURE 2 and shows it from another perspective  
18 to aid in the teaching the operation of the detailed embodiment of this specification. Thus,  
19 FIGURE 3 shows IL1 201 in more detail as having a section 201A containing program  
20 instructions and a section 201B containing hint instructions. That is, each row of  
21 instructions in IL1 201 has its leftmost part in section 201A for containing program  
22 instructions, and its rightmost part in section 201B for containing a hint instruction at the  
23 end of each row. Section 201A operates as an Instruction Cache (I-cache) of the type  
24 described in the incorporated US Patent 6598152, granted July 22, 2003 for increasing the  
25 overall prediction accuracy for multi-cycle branch prediction processes and apparatus for  
26 enabling quick recovery in generating new prediction for the BHT.



1 As previously mentioned, the address in IFAR 203 selects a row of program instructions in  
2 IL1 201 and an associated BHT row of prediction fields in BHT 204. FIGURE 3 illustrates  
3 the IFAR selected BHT row (i.e. also herein called the current BHT entry) being outputted  
4 to an "eight prediction" register 308, and the IFAR selected IL1 row (i.e. a group of 8  
5 program instruction fields) being outputted to an "eight program instructions" register  
6 309. Each branch instruction field in register 309 has an associated branch prediction field  
7 at a corresponding location in register 308. The associated branch prediction field is only  
8 used if the corresponding branch instruction field contains a conditional branch  
9 instruction. Hence, the associated branch prediction field is not used if the corresponding  
10 instruction field contains a non-branch instruction.

11 The "branch taken / not taken" state of each branch prediction bit in register 308 (when  
12 associated with a corresponding conditional branch instruction in register 309) is generally  
13 determined by the most-recent execution of that instruction in the current program. The  
14 correctness of this branch prediction is checked by branch execution logic 209,216,217A)  
15 after the IFAR selection of the corresponding branch instruction in the program.  
16 Whenever the check by branch execution logic (209,216,217A) finds a BHT prediction is  
17 correct, the last predicted execution path continues to be followed in the program without  
18 interruption. But when execution logic (209,216,217A) finds a BHT bit prediction is  
19 wrong, the wrong path has been followed in the program, the correct path must be found  
20 and followed, and the execution results of the wrong path are discarded. Thus, when logic  
21 (209,216,217A) finds the currently BHT prediction bit is wrong, the correct setting for that  
22 BHT bit also is determined, and the state of that BHT bit is changed to its correct state.  
23 The target address of the executed branch instruction is then known and is determinative  
24 of the location in the program execution from which the incorrect path began and is the  
25 beginning of the correct new execution path in the program.

26 This manner of operation for re-setting the execution path when a wrong prediction is  
27 detected by logic (209,216,217A) is described and claimed in the incorporated US Patent  
28 6598152, granted July 22, 2003. In more detail regarding this incorporated specification,

1 whenever a new row of instructions was fetched into its instruction cache (I-cache) from a  
2 storage hierarchy, these newly fetched instructions overlay and replace any and all  
3 instructions previously stored in that I-cache row. In this prior system, the BHT entry  
4 associated with that I-cache row are not replaced in the BHT when the associated IL1 row  
5 received the newly fetched instruction line. Thus, whatever pre-existing prediction states  
6 exist in the BHT entry (determined by execution of the replaced instructions in the row no  
7 longer in the I-cache) are then used as the branch predictions for the new unrelated branch  
8 instruction(s) newly fetched I-cache row and overlaying the corresponding that were used  
9 to generate those BHT bit states. When any BHT prediction bit is used and then later  
10 found incorrect by execution logic (209,216,217A), the bit is belatedly rewritten in its  
11 BHT location to correct it. The penalty for incorrectness of any BHT bit prediction is the  
12 loss of all execution results obtain from instructions executed in the wrong execution path  
13 and the time taken for such wrong executions. Hence for each BHT bit correction, many  
14 unneeded instructions may have been selected and executed, wasting many instruction  
15 selection and execution cycles which detract from the required program execution and  
16 decrease the program execution speed.

17 The rate of incorrect predictions is decreased by this invention enabling recent branch  
18 history lost in the prior art operation (when instructions are replaced in an instruction  
19 cache) to be retained in hint instructions and reused. The subject invention increases the  
20 likelihood of the associated BHT prediction field being correct for a I-cache row of  
21 instructions re-fetched into the instruction cache - by enabling the saving and reuse of the  
22 prediction fields associated with overlaid rows of instructions in an I-cache whenever that  
23 row of instructions is refetched during later execution of a program..

24 The top of the storage hierarchy in FIGURE 2 is herein called "level 1" in the storage  
25 hierarchy of the system and contains the instruction cache IL1 201 and a data cache  
26 (D-cache) 221. They respectively provide the instructions and data to the central processor  
27 for execution by the current program. The next level in this hierarchy is called "level 2"  
28 which provides the instruction lines and data to the level 1 caches, and herein is provided

1 by the L2 cache 212 which contains both instructions and data. It provides instruction  
2 lines to IL1 201 and lines of data to D-cache 221 in response to demands (misses) by the  
3 level 1 caches. L2 cache 212 obtains its instructions and data from the main memory of the  
4 computer system in the conventional manner of storage hierarchies.

5 The L2 cache of this invention has the unique structure for containing hint instructions  
6 which is not found in the prior art.

7 In IL1 201 (see FIGURE 6) and in the L2 cache with hint extensions (see FIGURE 7), the  
8 program instructions and the hint instruction are stored in predetermined locations in each  
9 of the cache entries to distinguish the program instructions from the hint instruction stored  
10 in the same cache entry. Thus the left part of each IL1 and L2 cache row contains space  
11 for storing a line of program instruction, and the right part of the row contains space for  
12 storing a hint instruction or a NOP instruction. The hint instruction locations in both the  
13 IL1 and L2 caches are initialized to contain NOP instructions, which are overlaid whenever  
14 a hint instruction is to be stored into the cache entry.

15 Thus initially during program execution, NOP instructions exist in the hint instruction  
16 locations in the IL1 and L2 caches. When an initial miss occurs for a program instruction  
17 line in both the IL1 cache entry, and in the L2 cache, the line of program instructions  
18 (containing the requested instruction) is fetched from system main storage into that line  
19 location in the L2 cache, and also into the IL1 cache entry. Later during the program  
20 execution, the space occupied by this IL1 cache entry may be needed for a new line of  
21 program instructions which maps to this same IL1 cache entry during execution of the  
22 program. Before the new line is allowed to be stored into this IL1 cache entry, the existing  
23 line in the IL1 cache entry is replaced in the IL1 cache and a hint instruction is stored into  
24 the L2 cache entry having the copy of the replaced instruction line with the hint instruction  
25 generated for the BHT entry of the replaced instruction line.

1 In FIGURE 2A, each hint instruction is generated in the detailed embodiment by “hint  
2 instruction generation” circuits in the hint processor when required during the program  
3 instruction executions. The detailed embodiment uses “hint instruction execution” circuits  
4 in the hint processor to execute each hint instruction when required during the program  
5 instruction executions. Alternatively to having separate hint processor hardware circuits,  
6 the same hint processor generation and execution functions may be provided by having  
7 these functions micro-coded as subprocesses in the central processor executing the  
8 program.

9 The general operation of the IL1 cache with concurrent hint instruction suboperations is  
10 done in FIGURES 2 and 3 as follows: When the central processor in FIGURE 2 needs to  
11 select an instruction, the IL1 cache row is selected by IFAR bits 48:58 (i.e. the current  
12 instruction address is in IFAR). If that row contains the IFAR addressed instruction, it is  
13 accessed to provide an IL1 hit. If that instruction is not found therein, an IL1 miss occurs.  
14  
15 An IL1 cache miss may occur under two different conditions: (1) The valid bit in the  
16 associated IL1 directory entry may be indicating the invalid state, which will cause a cache  
17 miss. (2) When that valid bit is indicating a valid state, a cache miss occurs if the current  
18 IFAR address bits 0-47 do not match the current address in the “address of the first  
19 instruction” field in the associated IL1 directory entry.

20 If an IL1 cache miss occurs for reason (1), i.e. because the directory valid bit indicates the  
21 invalid state, no valid instruction line exists in this IL1 cache entry and a new instruction  
22 line may immediately be fetched from the L2 cache and copied into that IL1 cache row.  
23 The hint instruction associated with the L2 cache copy of the line is copied into the hint  
24 instruction location in the same IL1 row. The form of the copied hint instruction is the  
25 form found in the copied L2 cache row, which is either 100 or 109 in FIGURE 1.

26 However, if an IL1 cache miss occurs because of reason (2), i.e. the directory valid bit  
27 indicates the valid state when the IL1 directory entry’s “address of the first instruction”

1 field does not match the current IFAR address bits 0-47, a valid instruction line exists in  
2 the IL1 cache row with an associated hint instruction, and the hint instruction must be  
3 castout to the L2 cache row containing a copy of the IL1 instruction line before it is  
4 overlaid by a hint instruction associated with the instruction line being fetched. This L2  
5 cache row is located by using the "address of the first instruction" field in the associated  
6 IL1 directory entry.

7 It will be recalled that current programs are comprised of read-only code which is not  
8 changed during execution of a program. Therefore the read-only program instructions in  
9 an existing line in IL1 201 do not need any castout operation (as is required for data  
10 changed by the program instructions in D-cache 221). Therefore, no castout is required for  
11 an IL1 line of program instructions about to be overlaid, since the line can be later  
12 obtained from a corresponding line in some level of the system storage hierarchy. A line of  
13 program instructions in an IL1 cache entry usually has a copy in a corresponding L2 cache  
14 entry, and the corresponding L2 cache entry may have copies at other levels of the storage  
15 hierarchy.

16 Hint instructions are generated and written into the IL1 and L2 cache rows by the hint  
17 instruction generation process when an instruction hit occurs in IL1. A hint instruction  
18 100 is generated and written into the hint instruction location in an associated IL1 row by  
19 the hint processor 206. This hint instruction generation process uses the current IFAR  
20 address and the associated line of program instructions to generate the fields in each hint  
21 instruction.

22 When a valid instruction line exists in the IL1 row having a miss, its associated hint  
23 instruction is executed by the hint processor 206 concurrent with its castout to its L2 cache  
24 row and while the newly fetched line of instructions is being written in the IL1 cache entry  
25 to overlay the associated line. The newly fetched hint instruction (from the IFAR  
26 addressed L2 cache row) is written into the hint instruction location, overlaying the  
27 executed hint instruction in that location.

1 It is to be noted that on any IL1 cache miss, the replacement new line of instructions is  
2 obtained from a different L2 cache entry than the L2 cache entry containing a copy of the  
3 replaced IL1 cache line causing the miss. It is further to be noted that branch instruction  
4 distribution in the replacement line may be independent of the branch instruction  
5 distribution in the replaced line. This has implications in the content of their BHT  
6 prediction values by indicating that each has a BHT content independent of the other.

7 The L2 cache is generally much larger than the IL1 cache and has many time more entries  
8 than the IL1 cache. The L2/IL1 entry ratio is preferably a power of two. In the described  
9 embodiment a ratio of 32 ( $32=2^{**5}$ ) is used. A small L2 cache may have twice the number  
10 of entries of the IL1 cache. An expected common occurrence during the IL1 cache misses  
11 for many IL1 cache entries is to have a replacement sequence for an IL1 cache entry which  
12 alternates between two different L2 cache instruction lines, which are respectively  
13 associated with two different hint instructions. These two instruction lines may have one or  
14 more branch instructions at different locations, and/or one or more branch instructions at  
15 the same location within their respective lines. This different branch instruction  
16 distribution characteristic can affect their respective BHT values during the operation of  
17 this invention.

18 A hint instruction stored in the L2 cache enables the current program to refetch the  
19 associated line from the L2 cache and restore the associated BHT prediction bits to the  
20 most recent prediction state(s) for any branch instructions in the line without disturbing  
21 the prediction states for any non-branch bit positions in the BHT entry. The implication of  
22 this is that the undisturbed states of the non-branch positions may continue to represent  
23 the latest predictions for any branch instruction(s) in an alternate instruction line when it  
24 is not stored in the IL1 row to which it maps. These BHT bit predictions for the  
25 non-branch positions have the advantage of not needing to be regenerated when the  
26 alternate line for which they were generated is later refetched into that IL1 row; whereby if

1 their states were disturbed it would increase the chance of selecting one or more wrong  
2 execution paths when the alternate line is again written in that IL1 cache row.

3 In this manner, the BHT prediction bit states for branch mask positions in the hint  
4 instructions stored in the L2 cache provide “hints” of the most recently used  
5 “taken/non-taken” branch state of each conditional branch instruction in their associated  
6 lines of instructions, whereby the mask indicated positions have a greater than 90% chance  
7 of providing correct predictions, instead of merely the 50% chance of providing a correct  
8 prediction if they were represented by the BHT values for the last instruction line in that  
9 IL1 cache entry.

---

10 In this manner, the hint instructions can restore the BHT bits for the branches in refetched  
11 IL1 lines to the prediction states most likely to correctly predict the branch outcomes.  
12 Thus the result of using the hint instructions of this application is to save processor  
13 execution time that would otherwise be lost in executing unnecessary instructions in  
14 wrongly selected execution paths in a program due to using incorrect BHT bit states for  
15 replaced IL1 lines.

16 FIGURE 7 shows the form of the L2 cache 212 and its directory 211 in the described  
17 embodiment. IFAR address bits 43 through 58 (43:58) are used as an index to locate and  
18 to associated a L2 cache entry and its corresponding L2 directory entry. Each L2 directory  
19 entry contains a “type bit” for indicating whether the addressed cache row contains an  
20 instruction line (I) or a data line (D). For example, type “1” may indicate a line of  
21 instructions, and type “0” may indicate a line of data words. Each L2 directory entry also  
22 contains the “address of the first instruction” in its associated line and a valid bit to  
23 indicate if the addressed line is valid.

24 The IL1 cache and the L2 cache used in the detailed flow diagrams in FIGURES 8 - 13 are  
25 shown in FIGURES 6 and 7, in which the IL1 cache is a dedicated instruction cache which  
26 only contains instructions, which in this specification can have two types of instructions

1 stored therein: "program instructions" and novel "hint instructions". There also is a IL1  
2 data cache 221 which contains the data accessed by the operands in the instructions  
3 executed from the IL1 instruction cache. This invention may also use a unified IL1 cache  
4 (not shown) containing both instructions and data.

5 In the detailed embodiment, a unified L2 cache is shown and used; it is a unified cache  
6 because it contains both instructions and data. Data cache operations are not used and are  
7 not needed in explaining this invention being claimed in this specification. In the  
8 corresponding L2 directory entry an "I" or "D" indication in a predetermined field  
9 indicates whether the associated line contains instructions or data, when the valid bit is set  
10 to the valid state in that L2 directory entry.

11 Each L1 and L2 cache row has space for a line of instructions and space for an associated  
12 hint instruction; the hint instruction space is in a predetermined location in each row,  
13 which may be anywhere in its row but is shown herein at the end of its line of instructions.  
14

15 Other tag bits (not shown) may also be included in each directory entry, for example, an L2  
16 directory entry containing a "D" indicator may also contain a "change bit" (not shown) to  
17 indicate if the data in the corresponding L2 cache entry has been changed since it was  
18 received by that L2 cache entry, whereby a castout of the contained data line need only be  
19 done if the data is indicated as having been changed. An "I" indication in a L2 directory  
20 entry does not need any "change bit" because the program instructions are not changeable  
21 in any cache entry.

22 Program instructions and data are fetched from the system storage hierarchy to the L2  
23 cache entries in response to an L2 cache miss. Program instructions are fetched from L2  
24 cache entries to IL1 cache entries in response to an IL1 cache miss.

25 However, only changed data in the L2 cache is castout to the system storage hierarchy  
26 when the data is to be replaced in an L2 cache entry. No castout is done for program



1 instructions, because all program instructions are presumed to be readonly and  
2 unchangeable in both the IL1 and L2 caches.

3 A line of program instructions may remain valid in the L2 cache entry as long as its L2  
4 cache space is not needed for other program instructions. The mask-located prediction bits  
5 in any BHT field in the L2 hint instruction remain usable as long as its associated line of  
6 program instructions is valid in the L2 cache. A BHT entry may later be restored by a hint  
7 instruction when the associated line of program instructions is later retrieved from a valid  
8 L2 cache entry having a hint instruction. The restored BHT prediction bits in a BHT entry  
9 have the prediction values existing when their hint instruction was generated at the time of  
10 the last hit in the line in a IL1 cache entry. The restored prediction states of the BHT bits  
11 provide "hints" as to the most likely taken, or not-taken, path from a branch instruction in  
12 the line of program instructions.

13 FIGURE 2A shows a detailed embodiment of hint processor hardware logic sub-circuits  
14 which are preferably located in the same semiconductor chip having the circuits used for  
15 processing the program instructions using the hint instructions. The hint processor is  
16 shown in two parts: a "hint instruction generation" part on the right of the vertical dashed  
17 line, and a "hint instruction execution" part on the left of the vertical dashed line.

18 In the hint processor in FIGURE 2A, the "hint instruction generation" circuits have a BHT  
19 hint write register 241 into which are loaded IFAR address bits 48:58. These address bits  
20 are also received in the eleven-bit "bht index" field having locations 5-15 in a BHT hint  
21 register 242. The hint instruction operation code is internally provided into its first four  
22 bit locations 0-4 comprising the "Ld\_bht\_op" field. Concurrently, all program instructions  
23 (up to 8 instructions comprising the instruction line in the selected IL1 cache entry) are  
24 copied to "branch mask creation logic" register 243, from which a "branch mask" field is  
25 formed in register 242. To form the mask, a "1" bit is stored in the branch mask field to  
26 locate each branch instruction in the line, and a "0" bit is stored in the branch mask field to  
27 locate each non-branch instruction in this field. Thus, in the branch mask field each bit

1 positions in the mask corresponds to the position of its represented program instruction in  
2 the line. The “bht\_bits” field at bit positions 24-31 in register 242 receives the bits in the  
3 BHT field located by the current IFAR address bits 48:58.

4 The content of register 242 is outputted to a hint instruction location in the IL1 cache entry  
5 located by IFAR bits 48:58 in register 241 when a new hint instruction is required by  
6 operation 907 in the process of FIGURE 9.

7 The “hint instruction execution” circuits of the hint processor in FIGURE 2A are used by  
8 the operation 822 in the process shown in FIGURE 8. This operation restores the bits in  
9 the current BHT entry for the branch instructions in a newly refetched line of instructions.  
10 Then, the hint instruction is fetched from the L2 cache to the IL1 cache and is executed by  
11 the “hint instruction execution” circuits of the hint processor in FIGURE 2A. The  
12 execution begins when the hint instruction is transferred into hint instruction register 231  
13 in the hint processor in FIGURE 2A. Concurrently, the associated BHT entry (eight bits  
14 located by the current IFAR bits 48:58) is copied to the “curr\_bht register 232. The  
15 “branch mask” field in bits 16-23, and the “bht-bits” field in register 231 are outputted to  
16 “new BHT creation logic” circuits 238, which outputs its created BHT value to a  
17 “new\_bht” register 239, from which it is written in the BHT field located by IFAR bits  
18 48:58 to overlay the current BHT entry in the BHT. Generally, the resultant BHT is a  
19 modification of the BHT received by the “curr\_bht register 232.

20 FIGURE 2B shows the circuit logic for bit position, n, within the “new BHT creation logic”  
21 circuits 238. Bit position n is duplicated for each of the eight BHT bit positions, 0 through  
22 7 comprising each BHT. Only one of the n bit positions may be changed at a time, and it is  
23 the bit position that is selected by the current IFAR address. The circuits for BHT bit n  
24 comprise two logical AND gates 251 and 252 having their outputs connected to an OR  
25 circuit 254, which provides the “new\_bit (n)” output that is written into the BHT at the  
26 current IFAR selected I-index. Thus, gate 251 receives the “bht\_bits(n)” bit in the  
27 “bht\_bits” field. Gate 252 receives “curr\_bht(n)” bit in the “curr\_bht” field. Gate 251 is

1 enabled by bit  $n$  in the “branch mask” field, called “branch\_mask( $n$ )”. Gates 251 and 252  
2 are alternately controlled by bit  $n$  in the “branch mask” field, wherein “branch\_mask( $n$ )”  
3 enables gate 251 and its inverted value outputted by inverter 253 disable gate 252 when  
4 gate 251 is enabled, and visa-versa. The eight bit content in the “new\_bht” register 239  
5 provides the output value written into the currently addressed BHT entry.

6 Having a L2 cache support two or more L2 lines simultaneously having copies in the IL1  
7 cache requires the L2 cache size to be at least twice as large as the IL1 cache. The L2/IL1  
8 ratio is the ratio of the number-of-L2-cache entries to the number-of-IL1 cache entries. In  
9 the detailed embodiment, the L2/IL1 ratio is a power-of-two ratio. When this ratio is two  
10 or more, it enables the L2 cache to simultaneously contain a copy of a current IL1 line, and  
11 a copy of a IL1 replacement line for the same IL1 cache entry. It is advantageous to make  
12 the L2 cache have several times the number of IL1 cache entries, in order to reduce the L2  
13 cache line thrashing caused by L2 cache misses which can delay the IL1 cache operations,  
14 when new lines of program instructions must be obtained from the system storage  
15 hierarchy. Thus at a minimum, the L2 cache should have at least twice the number of  
16 entries in the IL1 cache for a minimum ratio of two.

17 In the detailed embodiment, a L2/IL1 ratio of 32 ( $32=2^{**5}$ ) is used, which allows up to 32  
18 different L2 entries to map to each IL1 entry in the illustrated IL1 cache, which has 2048  
19 IL1 cache entries ( $2^{**11} = 2048$ ). These 11 bits are represented by bit positions 48:58 in  
20 any 64 bit address, and these bits 48:58 map into the IL1 cache the program address for a  
21 line of instructions, and the remaining high-order bits 0:47 of the 64 bit address are placed  
22 in the IL1 cache directory to identify the 64 bit address. To map any memory address into  
23 the IL1 cache, the 11 bits 48:58 in the 64 bit address are used as an index into the IL1 cache  
24 to select the IL1 cache entry. The remaining high-order bits 0:47 of the 64 bit address are  
25 placed in the IL1 cache directory to identify the 64 bit address in the IL1 cache directory  
26 entry at the same index (i.e. bits 48:58) as is used to locate the IL1 cache entry.

1 The L2 cache in the detailed embodiment has 65385 L2 cache entries ( $65386 = 2^{16}$ ),  
2 whereby  $65386/2048=32$  (which is the L2/IL1 size ratio). To map any 64 bit memory  
3 address into the L2 cache, its 16 bits 43:58 are used as an index into the L2 cache to select  
4 the L2 cache entry. The remaining high-order bits 0:42 of the 64 bit address are placed in  
5 the corresponding L2 cache directory entry located therein at the same index (i.e. bits  
6 43:58) as is used to locate the associated L2 cache entry to identify the same 64 bit address  
7 in that L2 cache directory entry. Thus, any 64 bit address may be mapped into the L2  
8 cache at L2 index 43:58 having its high-order bits 0-42 placed in the corresponding L2  
9 cache directory entry at this same index 43:58; and this same 64 bit address may be  
10 mapped into the IL1 cache at IL1 index 48:58 having its high-order bits 0-47 placed in the  
11 corresponding IL1 cache directory entry at this same index 48:58

12 Using these IL1 and L2 cache sizes, the memory address of the current IL1 line (to be  
13 replaced) is identified by IFAR bits 0-47 in the current IL1 directory entry located in the  
14 IL1 cache at the IL1 index determined by bits 48:58 of the IFAR address. The current IL1  
15 line (being replaced in IL1) has a copy in a L2 cache entry located in the L2 cache located  
16 by the address identified in an "address of the first instruction" field in this IL1 directory  
17 entry. The replacing line in the L2 cache has its copy is located at IFAR index 43:58 and its  
18 L2 directory entry contains bits 0:42 of this same memory address. A hint instruction is  
19 executed during the IL1 line replacement process, as the hint instruction is fetched from  
20 the L2 cache row, to modify the BHT to provide the best available BHT predictions for the  
21 branch instructions in the newly fetched line. A new hint instruction is generated each  
22 time an instruction hit is obtained in the line to refresh the hint instruction stored in the  
23 IL1 row to insure it has the latest predictions provided in the BHT for the branch  
24 instructions in the line.

25 The hint instructions enable a program to most efficiently perform its instruction  
26 executions. The avoidance of mispredictions by this invention avoids aborting execution  
27 selections in the processor's instruction execution pipeline where the branch instruction  
28 executions are belated checked and found to be incorrect due to executing mispredicted

1 branch instructions. Mispredictions cause much additional program delay due to  
2 additional instruction executions caused by backtracking the execution stream to correct  
3 mispredicted execution paths, requiring additional fetches of lines of instructions from the  
4 IL1 cache in a program that significantly slow the execution of the program. This invention  
5 can avoid most of the mispredicted target instruction delays, speeding up the execution of  
6 any program.

7 **Detailed Description of Processes and Subprocesses used by the detailed Embodiment:**

---

8 The process in FIGURE 8 is entered at operation 802 when program execution is started in  
9 the processor. Then operation 804 sets the processor's IFAR (instruction fetch address  
10 register) to the address of the first instruction in the program and start execution of the  
11 program. The processing performed in FIGURE 8 is concerned with hint instruction  
12 generation and use during a processor's selection and execution of program instructions in  
13 an IL1 instruction cache 201 utilizing BHT branch predictions, and using an L2 cache 212  
14 storing hint instructions during the execution of the program.

15 The next operation 806 uses the current IFAR address bit positions 48:58 as an IL1 index  
16 to locate a line of instructions in an entry in the IL1 directory 202. It is to be noted that  
17 operation 806 may be enter on the initiation of a program, and is reentered in response to  
18 an IL1 cache miss which causes operation 806 to be reentered on a loop back from the last  
19 operation 822 in FIGURE 8.

20 The next operation 807 tests the validity bit in the located IL1 directory entry. The state of  
21 the valid bit is written into a processor storage field called "valid\_IL1\_entry" which is set  
22 to a "0" state by operation 808 when the no path is taken from the operation 807 test when  
23 it indicates the IL1 directory entry is in the "invalid" state.

1 If operation 807 finds it valid, the yes path to operation 809 is taken and the  
2 "valid\_IL1\_entry" is set to the "1" state, which indicates a valid line exists in the current  
3 IL1 entry. Then operation 809 determines if the current IFAR address has a hit or miss  
4 with this valid line, and the "address of the first instruction" field is read from the IL1  
5 directory entry to determine the main memory address of the IL1 entry to be overlaid. The  
6 "address of the first instruction" field contains the high-order bits 0:47 of the memory  
7 address for locating the corresponding (associated) instruction in the IL1 cache 201 entry  
8 located by the current IFAR address bit positions 48:58. The first (or next) instruction to  
9 be executed in the program in this IL1 entry is located by bits 59 through 61 (i.e. 59:61) of  
10 the current IFAR address (used as an index in the current line of program instructions in  
11 the currently accessed IL1 cache entry).

12 An IL1 cache hit (IL1 hit) is obtained when operation 807 finds the valid bit in the valid  
13 state, and the yes path is take from operation 809 when the "address of the first  
14 instruction" field compare equal with the current IFAR bits 0:47, causing the process to go  
15 to FIGURE 9 entry B which enters operation 901 as the next operation in the process. But  
16 if operation 809 finds an unequal compare, the no path is taken to operation 810.

17 When operation 807 finds the valid bit in the invalid state, and operation 808 sets the  
18 "valid\_IL1\_entry" field to 0, operation 810 is entered. Operation 810 accesses the L2  
19 cache directory entry located by an L2 index determined by the current IFAR bits 43:58.

20 Then, operation 812 is entered. Operation 812 tests the L2 cache entry for an L2 cache  
21 hit/miss indicated by an valid/invalid bit state in the L2 cache directory entry. If invalid,  
22 the L2 cache directory does not contain a copy of the required line of program instructions  
23 for the IL1 with an accompanying hint instruction, and operation 815 is entered.

24 But if operation 812 finds a valid L2 entry, the yes path is taken to operation 813 to  
25 determine if the valid L2 entry has a L2 hit or L2 miss. An L2 miss is determined if  
26 operation 813 finds the address of the first instruction in the L2 cache directory entry

1 mis-matches with the current IFAR bits 0-42. Then, the no path is taken to operation 814,  
2 which checks the state of the type bit in the same L2 directory entry. An L2 cache miss is  
3 then determined if operation 814 finds the D (data) type is indicated for the addressed L2  
4 cache entry, since an I (instruction) type is required for the addressed L2 cache entry if a  
5 cache hit occurs, which would allow the instructions in that line to be fetched to the IL1.  
6 However, the D type indication (L2 cache miss) requires that operation 815 be entered to  
7 use the IFAR address to fetch a line of instructions in the system main memory and store  
8 that line into the currently addressed L2 cache entry, and the corresponding L2 directory  
9 entry is validated by setting its type bit to the I state and its valid bit to the valid state.

---

10 Operation 815 also sets a NOP hint instruction 109 into the hint instruction field of the  
11 addressed L2 cache entry for the new L2 instruction line, which will be fetched into the IL1  
12 as a new IL1 instruction line. Then, operation 817 checks the valid state of the IL1  
13 directory entry (valid if the "valid\_IL1\_entry" field equals 1) to determine if the  
14 corresponding IL1 entry contains a valid IL1 cache line which is about to be replaced in  
15 the IL1 entry.

16 When operation 817 finds the "valid\_IL1\_entry" set to the "0" (indicating a invalid state  
17 for the IL1 entry), there is no IL1 line to be overlaid. Therefore the IL1 entry is in a  
18 condition to receive the new replacing instruction line from the L2 cache, since there is no  
19 current IL1 entry to replace, and the no path is taken to operation 822.

20 Then, operation 822 accesses the L2 cache row addressed by IFAR bits 43:58 and transfers  
21 it to the currently accessed IL1 entry; that row contains an instruction line having "eight  
22 program instructions", and a hint instruction. This hint instruction is also forwarded to  
23 hint instruction register 231 in the hint instruction processor 206 shown in detail in  
24 FIGURE 2A, which then executes the hint instruction newly written into the accessed IL1  
25 entry from the L2 cache entry. Also, the current BHT entry is replaced with a modified  
26 BHT entry generated in the hint processor 206, as explained herein for FIGURES 2A, 2B  
27 and 13.

1 However, if operation 817 finds the “valid IL1\_entry” set to the “1” (indicating a valid IL1  
2 entry will be replaced which does not match the current IFAR bits), the process then  
3 follows its yes path to operation 816 which assigns a “IL1\_hint\_wr\_addr” field in a  
4 predetermined storage location and stores in it the IL1 cache index of the hint instruction  
5 which is provided by current IFAR bits 48:58. Operation 817 also assigns a  
6 “IL2\_hint\_wr\_addr” field in another predetermined storage location to the copy of the line  
7 about to be replaced in the IL1 cache, and stores its L2 cache index, which is the  
8 concatenation of bits 43:47 in the “address of the first instruction” field of the IL1  
9 directory entry located by IFAR bits 48:58 (now stored in the “IL1\_hint\_wr\_addr” field).  
10 Then operation 816 accesses the L2 directory entry at the address stored in the  
11 “IL2\_hint\_wr\_addr” field, and goes to operation 818.

12 For finding the L2 line address of the line to be fetched, operation 816 determines the L2  
13 address for the current line in IL1 by assigning a “IL1\_hint\_wr\_addr” field in a  
14 predetermined storage location to receive the current entry’s IL1 index, which is set to  
15 IFAR bits 48:58.

16 For locating the L2 copy of the current IL1 entry about to be replaced (which locates where  
17 the castout hint instruction is to be stored in the L2 cache), operation 816 assigns an  
18 “IL2\_hint\_wr\_addr” field in another predetermined storage location, and this field  
19 receives an L2 cache index equal to the concatenation of bits 43:47 of the “Address of the  
20 first instruction” field of the IL1 directory entry located by IFAR bits 48:58 in the  
21 “IL1\_hint\_wr\_addr” field. Then operation 816 accesses the L2 directory entry at the  
22 address indicated in the “IL2\_hint\_wr\_addr” field, and goes to operation 818.

23 Operation 818 tests if this L2 entry is valid and if it contains a copy of the required IL1 line  
24 by comparing the “address of the first instruction” field in the L2 directory and the  
25 “address of the first instruction” field in the current IL1 directory entry. Furthermore,  
26 operation 818 also checks the “type” field in this L2 directory entry for the “I” state. If all



1 of these tests by operation 818 are successful, the instruction line being replaced in IL1 has  
2 a copy in the L2 cache, and the process takes the yes path to operation 820. Operation 820  
3 writes the hint instruction from the current entry in the IL1 cache (indexed in IL1 by the  
4 current IFAR bits 48:58) to the hint instruction field of the L2 cache entry (in the row  
5 located in the L2 cache by the current content of the IL2\_hint\_wr\_addr field).

6 However, if operation 818 is unsuccessful, there is no valid instruction line to be replaced in  
7 IL1 and it cannot have a copy in the L2 cache, and the process goes to operation 822.  
8 Operation 822 loads the currently addressed IL1 row from the currently accessed L2 cache  
9 entry by transferring the "eight program instructions" field and the hint instruction field  
10 from the L2 cache entry located by IFAR bits 43:58. This hint instruction is also  
11 forwarded to the hint instruction processor in FIGURE 2A, which then executes the hint  
12 instruction process shown in FIGURE 13, and the FIGURE 13 process operates in parallel  
13 with a continuation of the process in FIGURE 8.

14 The process in FIGURE 13 is entered at operation 1301 for testing during the current  
15 IFAR cycle if the received instruction is a hint instruction. If the test does not find a hint  
16 instruction, the process takes the no path to its exit. If a hint instruction is found by  
17 operation 1301, the process goes to operation 1302 to test if the hint instruction operation  
18 code is the ld\_bht\_op field, or a NOP field. If a NOP is found, the process goes from  
19 operation 1301 to the exit in FIGURE 13. If a ld\_bht\_op field is found by operation 1302,  
20 the BHT write update path is followed (it uses the triggers "wr\_en hold 1" 236 and "wr\_en  
21 hold 2" 237 in FIGURE 2A) to send an a hint instruction interpretation enable signal.

22 Then the next operation 1303 is performed, and it reads the BHT entry indexed by the  
23 bht\_index field in the current hint instruction, and copies it into the curr\_bht register 232.

24 Then, operation 1304 (using the hint instruction in register 231) generates a new BHT  
25 entry for being set in a "new\_bht" register. It uses logical AND/OR bit by bit functions as  
26 previously explained herein for FIGURE 2B, in which each of the respective bit n is

1 generated for the "new\_bht" register as: (the nth curr\_bht bit AND the inversion of the nth  
2 "branch\_mask" bit) OR (the nth bht\_bits bit AND the nth "branch\_mask" field in the  
3 hint instruction).

4  
5 Finally, operation 1305 stores the eight bit "new\_bht" field value in the BHT entry  
6 currently indexed by the content of the "bht\_index" field of the hint instruction. The  
7 process in FIGURE 13 then exits and goes to FIGURE 8 operation 806 to again read the  
8 IL1 directory entry indexed by IFAR bits 48:58. Then operation 807 again tests this same  
9 IL1 directory entry for validity; and since it has been made valid, the next operation 809  
10 sets the "valid\_IL1\_entry" to 1, and finds that now the current IFAR bits 0:47 matches the  
11 "address of the first instruction" field in the new content in the same IL1 directory entry.  
12 An IL1 hit then occurs and the process goes to FIGURE 9 entry point B.

13 Operation 901 is entered in Figure 9 at entry point B. At operation 901, the IL1 cache line  
14 is fetched into the "Eight Program Instructions" register 309, and the associated hint  
15 instruction into the "Hint Instructions" register 231. Next, the BHT entry indexed by the  
16 IFAR bits 48:58 is accessed, and its BHT prediction bits are fetched into the "Eight  
17 Predictions" register 308.

18 Then operation 903 uses the IFAR bits 59:61 to locate a "first instruction" in the "Eight  
19 Program Instructions" register 309 (Instructions before the "first instruction", if any, will  
20 be ignored).

21 The next operation 904 is tests if there is any branch instruction in the "Eight Program  
22 Instructions" register 309 at or after the "first instruction"? If "no", operation 906 is  
23 entered and designates a "fetch group" as the instructions from the "first instruction" to  
24 the end of register 309. Then, a "Predicted\_IFAR" field in logic 311 is set to the address of  
25 the next sequential instruction after the "fetch group", and the process goes to operation  
26 926.

1 But if operation 904 takes its "yes" path, the process performs operation 907, which  
2 generates a new hint instruction in the currently selected IL1 cache row. This is done by  
3 the hint processor 206 (in FIGURE 2A) filling its BHT Hint register 242 with the following:  
4 bits 0:4 with "ld\_bht\_op", bits 5:15 with IFAR bits 48:58, bits 16:23 with an 8-bit "branch  
5 mask" field containing a 1 in the positions where there is a branch and 0 in other positions,  
6 bits 24:31 with the 8-bit BHT prediction. Then the hint processor stores IFAR bits 48:58 in  
7 the BHT Hint Write Entry register 241, and operation 907 finally stores the content of the  
8 BHT Hint register in the IL1 Hint Instruction Cache entry indexed by BHT Hint Write  
9 Entry register 241.

---

10 Then the next operation 911 determines if any branch bit in the "Eight Predictions"  
11 register 308 (which in FIGURE 3 receives the last-outputted BHT field) indicates an  
12 unconditional branch predicted taken, or a conditional branch predicted taken? If the  
13 "yes" path is determined, operation 912 is entered and logic 311 in FIGURE 3 sets  
14 "Predicted\_IFAR" address to the target of the first of these branches and designates this  
15 branch as the "last instruction", and operation 921 is entered.

16 .

17 But if the "no" path is determined by operation 911, then operation 914 is entered and logic  
18 311 in FIGURE 3 sets "Predicted\_IFAR" address to the instruction next sequential to the  
19 last instruction fetched: and the last instruction in the "Eight Instructions" register 309 is  
20 designated as the "last instruction", and operation 921 is entered.

21 Operation 921 then forms the "fetch group" to contain all instructions between the "first  
22 instruction" and the "last instruction" determined in the "Eight Program Instructions  
23 register 309. For each branch instruction in the "fetch group", operation 926 obtains an  
24 invalid entry in the Branch Information Queue (BIQ) 313 in FIGURE 3, and FIGURE 4  
25 shows BIQ 313 in more detail. Then in BIQ 313, operation 921 sets the valid bit to 1 state  
26 in this BIQ entry, loads the address of the branch into an "Address of the branch" field  
27 401, loads the branch target address in the "Predicted address" field 402 if the branch is  
28 predicted taken or loads the next sequential address in the "Predicted address" field 402 if

1 the branch is predicted not-taken, and stores the n-th bit in the "Eight Predictions"  
2 register 308 in a "BHT bit" field 403 if the branch is at position "n" in the fetch group.  
3 Finally, operation 921 places the branch instruction in Branch Issue Queue 314 for its  
4 subsequent execution. Then the process goes to operation 926.

5 Operation 926 forwards the "fetch group" to Instruction Decode Unit (IDU) 208 shown in  
6 FIGURES 2 and 3 and performs the Instruction Decode and Dispatch process shown in  
7 FIGURE 11 (this is also described in previously-cited US Patent 6598152, granted July 22,  
8 2003. The process in FIGURE 11 may precede in parallel with the process in FIGURE 9.

9 When the process in FIGURE 9 is completed, the process goes to entry point C in FIGURE  
10 10.

11 When the process in FIGURE 11 is entered, operation 1101 is performed to determine if a  
12 "fetch group" was forwarded by the instruction fetch unit (IFU) and if it is the "fetch  
13 group" identified in the current IFAR cycle (i.e. addressed by the current IFAR setting). If  
14 the test by operation 1101 finds no "fetch group" has been forwarded for the current IFAR  
15 cycle, the "no" path is taken to the exit the process in FIGURE 11.

16 However if the test by operation 1101 finds the "fetch group" is for the current IFAR cycle,  
17 the "yes" path is taken to operation 1102, which is performed by IDU 208, which then  
18 forms one or more "dispatch groups" from the received "fetch group" following the rules  
19 of dispatch group formation. (These rules are: Not more than five instructions per group,  
20 At most one branch instruction in each dispatch group, and The fifth slot in the dispatch  
21 group is reserved for branch instructions only and if there is not enough instructions to fill  
22 all the slots in the dispatch group which have inserted NOPs).

23 Then operation 1103 obtains an invalid entry in the Global Completion Table (GCT) in 209  
24 shown in FIGURE 2 and fill its fields with the information for the dispatch group and  
25 validates the entry.

1 Finally, operation 1103 places each of the instructions in the “dispatch group” in the issue  
2 queue, and makes it available to the process shown in FIGURE 12 for operation 926.

3 The FIGURE 12 process is done by the Branch Issue Queue 314 and Branch Execution  
4 Logic unit 217A with outputs 316A and 316B shown in FIGURE 3. In FIGURE 12 the  
5 process performs Instruction issue and instruction execution operations, in which  
6 operation 1201 is entered. Operation 1201 determines if there is any valid Instruction in  
7 the Issue Queue for which all the operands are known? If “no”, the process waits one cycle  
8 and then again performs operation 1201 until a valid instruction is detected in the Issue  
9 Queue for which all operands are known.

10 Operation 1203 is entered from the “yes” path from operation 1201. Then, operation 1203  
11 forwards the detected Instruction to its proper execution unit 217A - 217D, which is one of  
12 the execution units shown in the Instruction Execution Units (IEU) 217 in FIGURE 2,  
13 which involves sending a branch instruction to the branch execution unit 217A, a load/store  
14 instruction to the load/store execution unit 217D, a fixed-point instruction to the fixed-point  
15 execution unit 217B, and a floating-point instruction to the floating-point execution unit  
16 217C. When the respective execution unit receives an instruction, it executes the  
17 instruction.

18 Operation 1203 forwards the instruction to its proper execution unit in the instruction  
19 execution unit 217 in FIGURE 2, and then operation 1204 executes the instruction. The  
20 process in FIGURE 12 then goes back to operation 1201 to repeat its operations for another  
21 valid instruction in the issue queue.

22 When operation 1203 forwards a conditional branch instruction to the branch execution  
23 logic 217A, it determines if the actual “branch taken/not taken” path is the same as the  
24 predicted “branch taken/not taken” path made by the BHT bit prediction for this  
25 instruction. If the actual and predicted are the same, the process in FIGURE 10 continues  
26 the predicted instruction stream. But if the determination finds they are not the same, then

1 the target instruction selected in the predicted instruction stream is in error, and the  
2 execution results of that branch target execution, and of all of its following instruction  
3 executions, must be flushed (eliminated) from the execution results for the current  
4 program, and they must be replaced by executing the instructions beginning with the  
5 actual target instruction determined by the actual execution of the wrongly predicted  
6 branch instruction.

7 In FIGURE 10, operation 1001 determines if the current instruction is being executed in  
8 the current cycle is a branch instruction. If no branch instruction is being executed, the  
9 program execution sequence is not affected; then the “no” path is taken to operation 1002,  
10 which occurs for most instructions in a program.. But if the currently executing  
11 instruction is a branch instruction, the “yes” path is taken to operation 1003.

12 When the “no” path is taken from operation 1001 to operation 1002, operation 1002  
13 determines if any non-branch flush signal has been received. Mostly non-flush signals are  
14 not received because the predictions are correct, and the “no” path is taken to operation  
15 1002 which sets the IFAR to the “predicted\_IFAR” address value. Then the subprocess in  
16 FIGURE 10 is ended, and the process goes to FIGURE 8 entry point A.

17 However, if the “yes” path is taken from operation 1005 to operation 1006, operation 1006  
18 sets IFAR to the non-branch flush address received. Then the subprocess in FIGURE 10 is  
19 ended, and the process goes to FIGURE 8 entry point A.

20 When a branch instruction is being executed, operation 1003 is performed using the  
21 Branch Information Queue (BIQ) hardware in FIGURE 4, and the operation reads the  
22 current BHT bit 403 and the Predicted Address 402 (for predicting the outcome of the  
23 currently executed branch instruction) in the current BIQ entry in BIQ 313. Then,  
24 operation 1003 determines if the branch instruction is mispredicted by finding if the valid  
25 bit 404 indicates the invalid state, or the actual target address is different from the  
26 predicted address 402. That is, the predicted and actual addresses are compared, and if

1 they do not have the same value, this branch instruction has a misprediction; then  
2 operation 1003 takes its “yes” path to operation 1007.

3 The usual case for operation 1003 is to find no misprediction (i.e. the compared predicted  
4 and actual addresses have the same value), and then the “no” path is taken to operation  
5 1004. Operation 1004 sets IFAR to the “Predicted IFAR” value, which is the address of the  
6 target instruction of this executed branch instruction. Then operation 1011 is entered, and  
7 the BIQ entry is released for this executed branch instruction by setting its BIQ valid bit  
8 404 to “0” state. The subprocess in FIGURE 10 is ended, and it goes to FIGURE 8 entry  
9 point A.

10 However, when the “yes” path from operation 1003 to 1007 is taken, a determination is  
11 made if the prediction by BHT bit 403 is correct. It is possible for the state of BHT bit 403  
12 to be correct and for a misprediction to nevertheless exist. If operation 1007 finds the BHT  
13 bit prediction is not correct, operation 1012 is entered. But if operation 1007 finds the BHT  
14 bit prediction is correct, operation 1017 is entered.

15 If the BHT bit prediction is correct, and operation 1017 is entered, then operation 1017 sets  
16 “Execution IFAR” to the target address of the branch instruction, and sets IFAR to the  
17 “Execution IFAR” value, and flushes all instructions from the instruction pipeline  
18 following the current branch instruction. Finally, operation 1021 releases the BIQ entry  
19 for the executed branch instruction by setting its valid bit to the “0” state. The process  
20 then goes to FIGURE 8 entry point A.

21 But if operation 1007 finds the BHT bit prediction is not correct, operation 1012 is entered  
22 to determine if the branch outcome is “taken”. If “taken”, operation 1014 sets “the  
23 “Execution IFAR” value to the target address of the branch instruction. If “not taken”,  
24 operation 1016 sets the “Execution IFAR” value to the value obtained by adding 4 to the  
25 “Address of the branch” field in the BIQ entry for the executed branch to generate the  
26 address of the next sequential instruction in the program..

1 When performed, each operation 1014 or 1016 enters operation 1018, which sets IFAR to  
2 the "Execution IFAR" value, and the execution results obtained for all instructions  
3 following the branch are flushed from instruction pipeline.

4 Then, operation 1019 sets a BHT\_write\_addr register 318 to the "address of the branch"  
5 field obtained from the BIQ entry for the executed branch. The BHT\_write\_data is set to  
6 1, if the branch outcome is "taken", else it is set to 0, and this value is written over the  
7 current BHT bit in the BHT to insure that it is corrected.

---

8 The next operation 1021 is then performed, and it releases the BIQ entry for the executed  
9 branch instruction by setting its valid bit to the "0" state. The process then goes to  
10 FIGURE 8 entry point A to repeat its operation which has been previously described  
11 herein..

12 While I have described the preferred embodiment of my invention, it will be understood  
13 that those skilled in the art, both now and in the future, may make various improvements  
14 and enhancements which fall within the scope of the claims, which follow. These claims  
15 should be construed to maintain the proper protection for the invention first disclosed  
16 here.

17 The invention claimed is:



1 Claim 15. (Currently amended) A branch prediction process for a computer  
2 system for improving branch prediction rate when using a branch history table,  
3 comprising:

4 determining if a program instruction processor (processor) has an access hit (hit) or access  
5 miss (miss) in an instruction cache (I-cache) when utilizing an instruction address (IFAR  
6 address) in attempting to select a program instruction for execution by the processor,

7 generating a hint instruction when the program instruction is a branch in response to a hit  
8 occurring during the determining operation, storing the hint instruction in association with  
9 a copy of an instruction line containing the program instruction in a storage hierarchy of  
10 the computer system, the hint instruction storing BHT prediction fields obtained from a  
11 copy of a current BHT entry associated with the instruction line when the hit occurs, and  
12 storing a branch mask in the hint instruction for locating an associated BHT field  
13 indicating the BHT field associated with the location of the program instruction in the  
14 instruction line, and

15 transferring the copy of the instruction line and associated hint instruction from the  
16 storage hierarchy to the I-cache in response to a miss occurring during the determining  
17 operation, and executing the hint instruction to restore a BHT prediction field in a current  
18 BHT entry to the state of a BHT field in the hint instruction located by the branch mask,  
19 and

20 the generating operation of generating hint instructions being performed by a hint  
21 processor operating in parallel with the program instruction processor, and

22 executing a hint instruction when the hint instruction is received in the I-cache by testing  
23 an operation code field in the hint instruction to determine if a completed hint instruction  
24 is indicated or if a no-operation state is indicated for the hint instruction, and continuing

1 the executing process only if a completed hint instruction is indicated by performing the  
2 following operations:

3 reading a BHT entry in the BHT located at an index determined by a bht\_index field in the  
4 hint instruction, and storing the BHT entry in a curr\_bht register,

5 logically ANDing an Nth bit in an inversion of the branch\_mask field in the hint instruction  
6 with an Nth bit in the curr\_bht register, where N is the bit position of the current  
7 instruction in the instruction line, and logically ANDing the Nth bit in a branch\_mask field  
8 with an Nth bit in a bht\_bits in the hint instruction,

---

9 logically ORing outputs of the two logical ANDing operations to provide an Nth bit output,  
10 and setting an Nth bit in a new\_bht register to the Nth bit output,

11 receiving without change in the new\_bht register at bit locations other than at the Nth bit  
12 location the bits in the curr\_bht register at corresponding bit locations other than the Nth  
13 location, and

14 setting the content of the new\_bht register into the current BHT entry in the BHT to  
15 restore the BHT entry to its last prediction state for the current instruction.

16 16. (Currently Amended) The branch prediction process for a computer system for  
17 improving branch prediction rates when using a branch history table as defined by claim  
18 15, further comprising:

19 performing all of the hint instruction operations in a hint instruction processor.

- 1   **17. (Currently Amended) The branch prediction process for a computer system for**  
2   **improving branch prediction rates when using a branch history table as defined by claim**  
3   **16, further comprising:**
- 4   **performing all hint instruction operations and all program instruction processor**  
5   **operations in a single semiconductor chip.**
-

1       **ABSTRACT OF THE DISCLOSURE**

2   **Apparatus and methods implemented in a processor semiconductor logic chip for**  
3   **providing novel "hint instructions" that uniquely preserve and reuse branch predictions**  
4   **replaced in a branch history table (BHT). A branch prediction is lost in the BHT after its**  
5   **associated instruction is replaced in an instruction cache. The unique "hint instructions"**  
6   **are generated and stored in a unique instruction cache which associates each hint**  
7   **instruction with a line of instructions. The hint instructions contains the latest branch**  
8   **history for all branch instructions executed in an associated line of instructions, and they**  
9   **are stored in the instruction cache during instruction cache hits in the associated line.**  
10   **During an instruction cache miss in an instruction line, the associated hint instruction is**  
11   **stored in a second level cache with a copy of the associated instruction line being replaced**  
12   **in the instruction cache. In the second level cache, the copy of the line is located through**  
13   **the instruction cache directory entry associated with the line being replaced in the**  
14   **instruction cache. Later, the hint instruction can be retrieved into the instruction cache**  
15   **when its associated instruction line is fetched from the second level cache, and then its**  
16   **associated hint instruction is also retrieved and used to restore the latest branch**  
17   **predictions for that instruction line. In the prior art this branch prediction would have**  
18   **been lost. It is estimated that this invention improves program performance for each**  
19   **replaced branch prediction by about 80%, due to increasing the probability of BHT bits**  
20   **correctly predicting the branch paths in the program from about 50% to over 90%. Each**  
21   **incorrect BHT branch prediction may result in the loss of many execution cycles, resulting**  
22   **in additional instruction re-execution overhead when incorrect branch paths are belatedly**  
23   **discovered.**